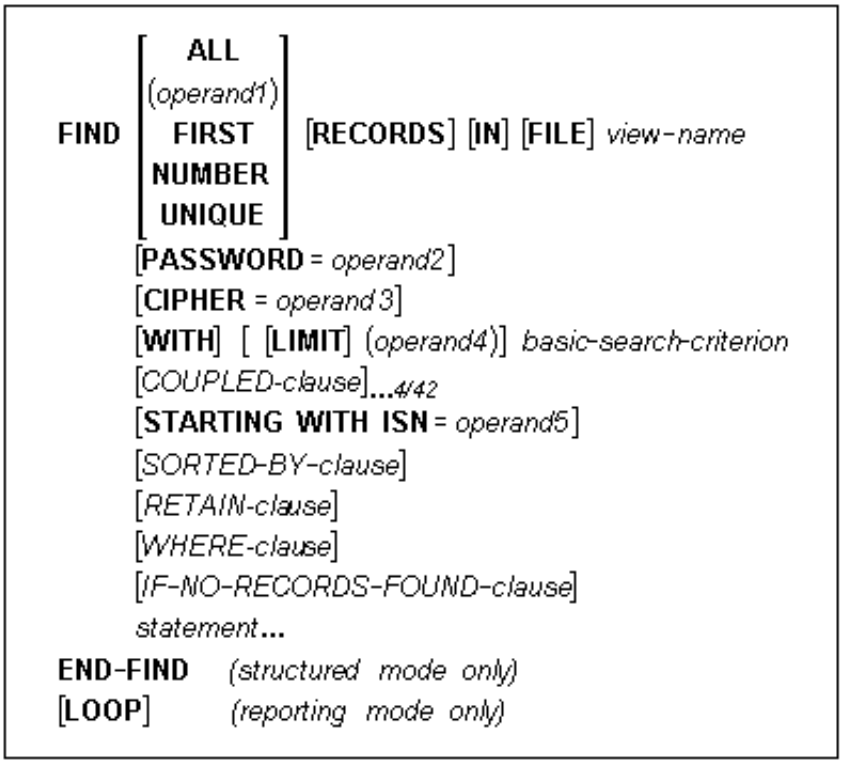
FIND



Operand	Possible Structure					Possible Formats											Referencing Permitted	Dynamic Definition
Operand1	C	S					N	P	I								yes	no
Operand2	C	S				A											yes	no
Operand3	C	S					N										yes	no
Operand4	C	S					N	P	I		B						yes	no
Operand5	C	S					N	P	I		B						yes	no

Related Statements: READ | HISTOGRAM

Function

The FIND statement is used to select a set of records from the database based on a search criterion consisting of fields defined as descriptors (keys).

This statement causes a processing loop to be initiated and then executed for each record selected. Each field in each record may be referenced within the processing loop. It is not necessary to issue a READ statement following the FIND in order to reference the fields within each record selected.

Considerations for DL/I Databases

When accessing a field starting after the last byte of the given segment occurrence, the storage copy of this field is filled according to its format (numeric, blank, etc.). The term segment occurrences should be substituted for the term records as used in this description of the FIND statement.

Considerations for SQL Databases

FIND FIRST as well as the PASSWORD, CIPHER, COUPLED and RETAIN clauses are not permitted.

FIND UNIQUE is not permitted. (Exception: On mainframe computers, FIND UNIQUE can be used for primary keys; however, this is only permitted for compatibility reasons and should not be used.)

The SORTED BY clause corresponds with the SQL clause ORDER BY.

The basic search criterion for an SQL-database table may be specified in the same manner as for an Adabas file. The term record used in this context corresponds with the SQL term `row`.

Considerations for VSAM Databases

The FIND statement is only valid for key-sequenced (KSDS) and entry-sequenced (ESDS) VSAM datasets. For ESDS, an alternate index for the base cluster must be defined.

Entire System Server Restrictions

FIND NUMBER and FIND UNIQUE as well as the PASSWORD, CIPHER, COUPLED and RETAIN clauses are not permitted.

Processing Limit - ALL/operand1

The number of records to be processed from the selected set may be limited by specifying *operand1* either as a numeric constant or as the name of a numeric variable enclosed in parentheses. ALL may be optionally specified and emphasizes that all selected records are to be processed.

If you specify a limit with *operand1*, this limit applies to the FIND loop being initiated. Records rejected for processing by the WHERE clause are not counted against this limit.

```
FIND (5) IN EMPLOYEES WITH ...

MOVE 10 TO #CNT(N2)
FIND (#CNT) EMPLOYEES WITH ...
```

For this statement, the specified limit has priority over a limit set with a LIMIT statement.

If a smaller limit is set with the LT parameter, the LT limit applies.

Notes:
If you wish to process a 4-digit number of records, specify it with a leading zero: (0nnnn); because Natural interprets every 4-digit number enclosed in parentheses as a line-number reference to a statement.
Operand1 has no influence on the size of an ISN set that is to be retained by a RETAIN clause.
Operand1 is evaluated when the FIND loop is entered. If the value of operand1 is modified within the FIND loop, this does not affect the number of records processed.

FIND FIRST, FIND NUMBER, FIND UNIQUE

These options are used to select the first record of a selected set (FIND FIRST), to determine the number of records in a selected set (FIND NUMBER), or to ensure that only one record satisfies a selection criterion (FIND UNIQUE).

These options are described in detail at the end of the FIND statement description.

view-name

The name of a view as defined either within a DEFINE DATA block or in a separate global or local data. In reporting mode, *view-name* may also be the name of a DDM.

PASSWORD Clause

The PASSWORD clause applies only for Adabas or VSAM databases. This clause is not permitted with Entire System Server.

The PASSWORD clause is used to provide a password (operand2) when retrieving data from an Adabas file which is password protected. If you require access to a password-protected file, contact the person responsible for database security concerning password usage/assignment.

If the password is specified as a constant, the PASSWORD clause should always be coded at the very beginning of a source-code line; this ensures that the password is not visible/displayable in the source code of the program. In TP mode, you may enter the PASSWORD clause invisible by entering the terminal command "%*" before you type in the PASSWORD clause.

If the PASSWORD clause is omitted, the password specified with the PASSW statement applies.

The password value must not be changed during the execution of a processing loop.

Example of PASSWORD Clause:

```
/* EXAMPLE 'FNDPWD': FIND (USING PASSWORD CLAUSE)
/************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
1 #PASS (A8)
END-DEFINE
/************************************************************
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:' #PASS (AD=N)
LIMIT 2
/************************************************************
FIND EMPLOY-VIEW
     PASSWORD = #PASS
     WITH NAME = 'SMITH'
 DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
/************************************************************
END
```



```
ENTER PASSWORD FOR EMPLOYEE FILE:
```

CIPHER Clause

The CIPHER clause only applies to Adabas databases. This clause is not permitted with Entire System Server.

The CIPHER clause is used to provide a cipher key (operand3) when retrieving data from Adabas files which are enciphered. If you require access to an enciphered file, contact the person responsible for database security concerning cipher key usage/assignment.

The cipher key may be specified as a numeric constant (8 digits) or the content of a user-defined variable with format/length N8.

If the cipher key is specified as a constant, the CIPHER clause should always be coded at the very beginning of a source-code line; this ensures that the cipher key is not visible/displayable in the source code of the program. In TP mode, you may enter the CIPHER clause invisible by entering the Natural terminal command "%*" before you type in the CIPHER clause.

The value of the cipher key must not be changed during the processing of a loop initiated by a FIND statement.

Example of CIPHER Clause:

```
/* EXAMPLE 'FNDCIP': FIND (USING CIPHER CLAUSE)
/************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 PERSONNEL-ID
1 #PASS (A8)
1 #CIPHER (N8)
END-DEFINE
/************************************************************
LIMIT 2
INPUT 'ENTER PASSWORD FOR EMPLOYEE FILE:   ' #PASS (AD=N)
    / 'ENTER CIPHER KEY FOR EMPLOYEE FILE: ' #CIPHER (AD=N)
/************************************************************
FIND EMPLOY-VIEW
     PASSWORD = #PASS
     CIPHER   = #CIPHER
     WITH NAME = 'SMITH'
 DISPLAY NOTITLE NAME PERSONNEL-ID
END-FIND
/************************************************************
END
```

```
ENTER PASSWORD FOR EMPLOYEE FILE:
ENTER CIPHER KEY FOR EMPLOYEE FILE:
```


WITH Clause

The WITH clause is required. It is used to specify the *basic-search-criterion* consisting of key fields (descriptors) defined in the database.

For Adabas files, you may use Adabas descriptors, subdescriptors, superdescriptors, hyperdescriptors, and phonetic descriptors within a WITH clause. On mainframe computers, a non-descriptor (that is, a field marked in the DDM with "N") can also be specified.

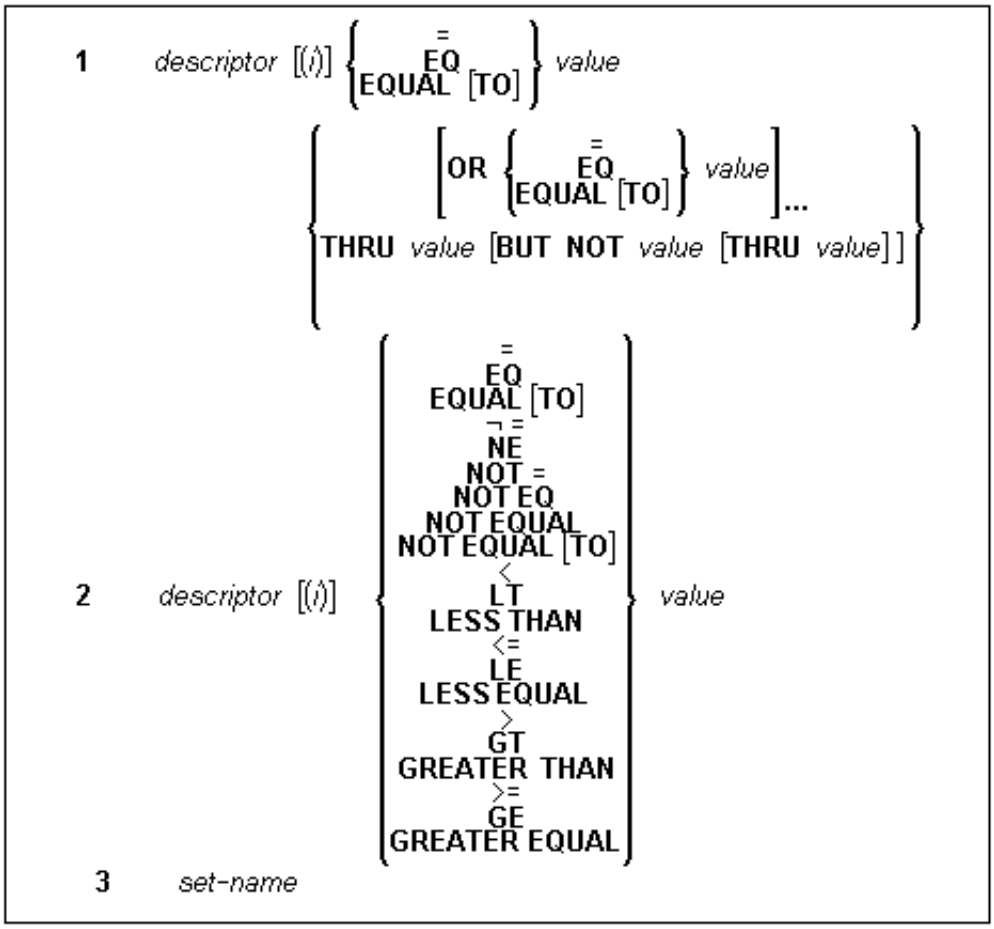
For DL/I files, you may only use key fields marked with "D" in the DDM.

For VSAM files, you may use VSAM key fields only.

The number of records to be selected as a result of a WITH clause may be limited by specifying the keyword LIMIT together with a numeric constant or a user-defined variable, enclosed within parentheses, which contains the limit value (*operand4*). If the number of records selected exceeds the limit, the program will be terminated with an error message.

Note:
If the limit is to be a 4-digit number, specify it with a leading zero: (0nnnn); because Natural interprets every 4-digit number enclosed in parentheses as a line-number reference to a statement.

Search Criterion for Adabas Files - basic-search-criterion



Operand	Possible Structure					Possible Formats												Referencing Permitted	Dynamic Definition
Descriptor		S	A			A	N	P	I	F	B	D	T	L				no	no
Value	C	S				A	N	P	I	F	B	D	T	L				yes	no
Set-name	C	S				A												no	no

descriptor

Adabas descriptor, subdescriptor, superdescriptor, hyperdescriptor, or phonetic descriptor. A field marked as non-descriptor in the DDM can also be specified.

i

A descriptor contained within a periodic group may be specified with or without an index. If no index is specified, the record will be selected if the value specified is located in any occurrence. If an index is specified, the record is selected only if the value is located in the occurrence specified by the index. The index specified must be a constant. An index range must not be used.

No index must be specified for a descriptor which is a multiple-value field. The record will be selected if the value is located in the record regardless of the position of the value.

value

Search value. The formats of the descriptor and the search value must be compatible.

set-name

Identifies a set of records previously selected with a FIND statement in which the RETAIN clause was specified. The set referenced in a FIND must have been created from the same physical Adabas file. *set-name* may be specified as a text constant (maximum 32 characters) or as the content of an alphanumeric variable. *set-name* cannot be used with Entire System Server.

Examples of Basic Search Criterion in WITH Clause:

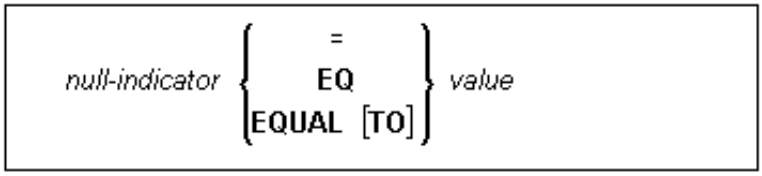
```
FIND STAFF WITH NAME = 'SMITH'
FIND STAFF WITH CITY NE 'BOSTON'
FIND STAFF WITH BIRTH = 610803
FIND STAFF WITH BIRTH = 610803 THRU 610811
FIND STAFF WITH NAME = 'O HARA' OR = 'JONES' OR = 'JACKSON'
FIND STAFF WITH PERSONNEL-ID = 100082 THRU 100100
                                  BUT NOT 100087 THRU 100095
```


Examples of Basic Search Criterion with Multiple-Value Field:

When the descriptor used in the basic search criterion is a multiple-value field, basically four different kinds of results can be obtained (the field MU-FIELD in the following examples is assumed to be a multiple-value field):

1. FIND XYZ-VIEW WITH **MU-FIELD = 'A'**
   This statement returns records in which *at least one* occurrence of MU-FIELD has the value "A".
2. FIND XYZ-VIEW WITH **MU-FIELD NOT EQUAL 'A'**
   This statement returns records in which *at least one* occurrence of MU-FIELD does *not* have the value "A".
3. FIND XYZ-VIEW WITH **NOT MU-FIELD NOT EQUAL 'A'**
   This statement returns records in which *every* occurrence of MU-FIELD has the value "A".
4. FIND XYZ-VIEW WITH **NOT MU-FIELD = 'A'**
   This statement returns records in which *none* of the occurrences of MU-FIELD has the value "A".

Search Criterion with Null Indicator - basic-search-criterion



Operand	Possible Structure					Possible Formats												Referencing Permitted	Dynamic Definition
Null-indicator		S							I									no	no
Value	C	S					N	P	I	F	B							yes	no

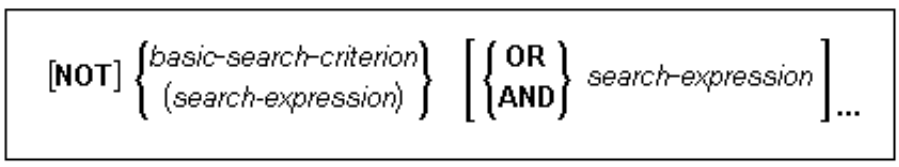
Possible *value* is "-1" (= the corresponding field contains no value) or "0" (= the corresponding field does contain a value).

Connecting Search Criteria (for Adabas Files)

Basic-search-criteria can be combined using the Boolean operators AND, OR, and NOT. Parentheses may also be used to control the order of evaluation. The order of evaluation is as follows:

1. **( )** Parentheses
2. **NOT** Negation (only for a *basic-search-criterion* of form [2]).
3. **AND** AND connection
4. **OR** OR connection

Basic-search-criteria may be connected by logical operators to form a complex *search-expression*. The syntax for such a complex *search-expression* is as follows:



Examples of Complex Search Expression in WITH Clause:

```
FIND STAFF WITH BIRTH LT 19770101 AND DEPT = 'DEPT06'
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                    AND (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH JOB-TITLE = 'CLERK TYPIST'
                    AND NOT (BIRTH GT 19560101 OR LANG = 'SPANISH')
```

```
FIND STAFF WITH DEPT = 'ABC' THRU 'DEF'
                    AND CITY = 'WASHINGTON' OR = 'LOS ANGELES'
                    AND BIRTH GT 19360101
```

```
FIND CARS WITH MAKE = 'VOLKSWAGEN'
                   AND COLOR = 'RED' OR = 'BLUE' OR = 'BLACK'
```

Descriptor - Key - Usage

Adabas users may use database fields which are defined as descriptors to construct basic search criteria.

Subdescriptors, Superdescriptors, Hyperdescriptors and Phonetic Descriptors

With Adabas, subdescriptors, superdescriptors, hyperdescriptors and phonetic descriptors may be used to construct search criteria.

- A subdescriptor is a descriptor formed from a portion of a field.
- A superdescriptor is a descriptor whose value is formed from one or more fields or portions of fields.
- A hyperdescriptor is a descriptor which is formed using a user-defined algorithm.
- A phonetic descriptor is a descriptor which allows the user to perform a phonetic search on a field (for example, a person's name). A phonetic search results in the return of all values which sound similar to the search value.

Which fields may be used as descriptors, subdescriptors, superdescriptors, hyperdescriptors and phonetic descriptors with which file is defined in the corresponding DDM.

Values for Subdescriptors, Superdescriptors, Phonetic Descriptors

Values used with these types of descriptors must be compatible with the internal format of the descriptor. The internal format of a subdescriptor is the same as the format of the field from which the subdescriptor is derived. The internal format of a superdescriptor is binary if all of the fields from which it is derived are defined with numeric format; otherwise, the format is alphanumeric. Phonetic descriptors always have alphanumeric format.

Values for subdescriptors and superdescriptors may be specified in the following ways:

- Numeric or hexadecimal constants may be specified. A hexadecimal constant must be used for a value for a superdescriptor which has binary format (see above).
- Values in user-defined variable fields may be specified using the REDEFINE statement to select the portions that form the subdescriptor or superdescriptor value.

Using Descriptors Contained within a Database Array

A descriptor which is contained within a database array may also be used in the construction of basic search criterion. For Adabas databases, such a descriptor may be a multiple-value field or a field contained within a periodic group.

A descriptor contained within a periodic group may be specified with or without an index. If no index is specified, the record will be selected if the value specified is located in any occurrence. If an index is specified, the record is selected only if the value is located in the occurrence specified by the index. The index specified must be a constant. An index range must not be used.

No index must be specified for a descriptor which is a multiple-value field. The record will be selected if the value is located in the record regardless of the position of the value.

Examples using Database Arrays:

The following examples assume that the field SALARY is a descriptor contained within a periodic group, and the field LANG is a multiple-value field.

```
1. FIND EMPLOYEES WITH SALARY LT 20000
```

(results in a search of all occurrences of SALARY)

```
2. FIND EMPLOYEES WITH SALARY (1) LT 20000
```

(results in a search of the first occurrence only)

```
3. FIND EMPLOYEES WITH SALARY (1:4) LT 20000    /* invalid
```

(a range specification must not be specified for a field within a periodic group used as a search criterion)

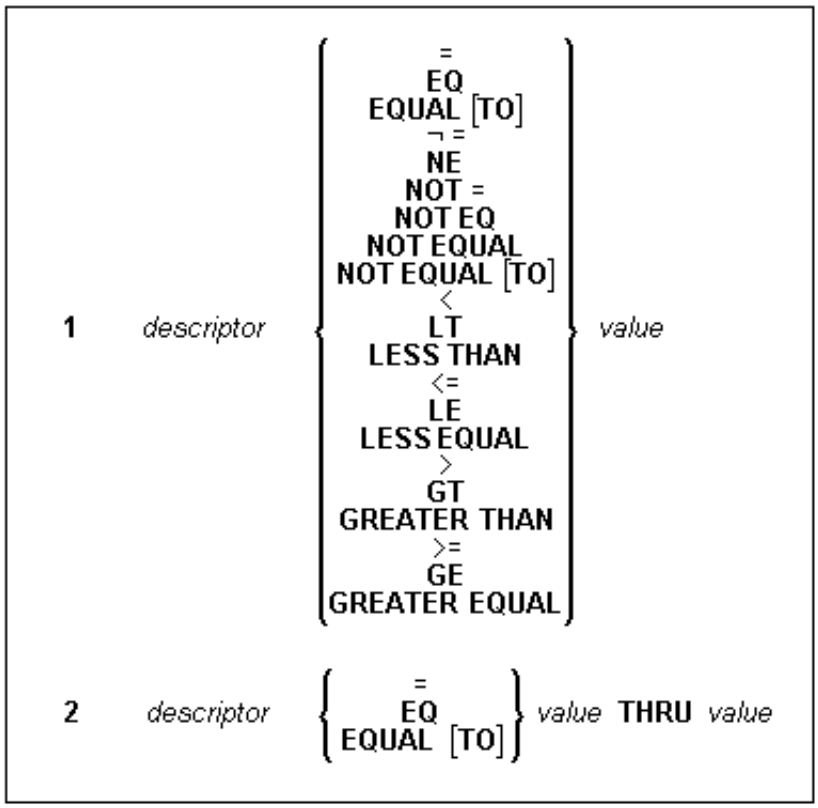
```
4. FIND EMPLOYEES WITH LANG = 'FRENCH'
```

(results in a search of all values of LANG)

```
5. FIND EMPLOYEES WITH LANG (1) = 'FRENCH'        /* invalid
```

(an index must not be specified for a multiple-value field used as a search criterion)

Search Criterion for VSAM Files - basic-search-criterion



Operand	Possible Structure					Possible Formats											Referencing Permitted	Dynamic Definition
Descriptor		S	A			A	N	P			B						no	no
Value	C	S				A	N	P			B						yes	no

descriptor

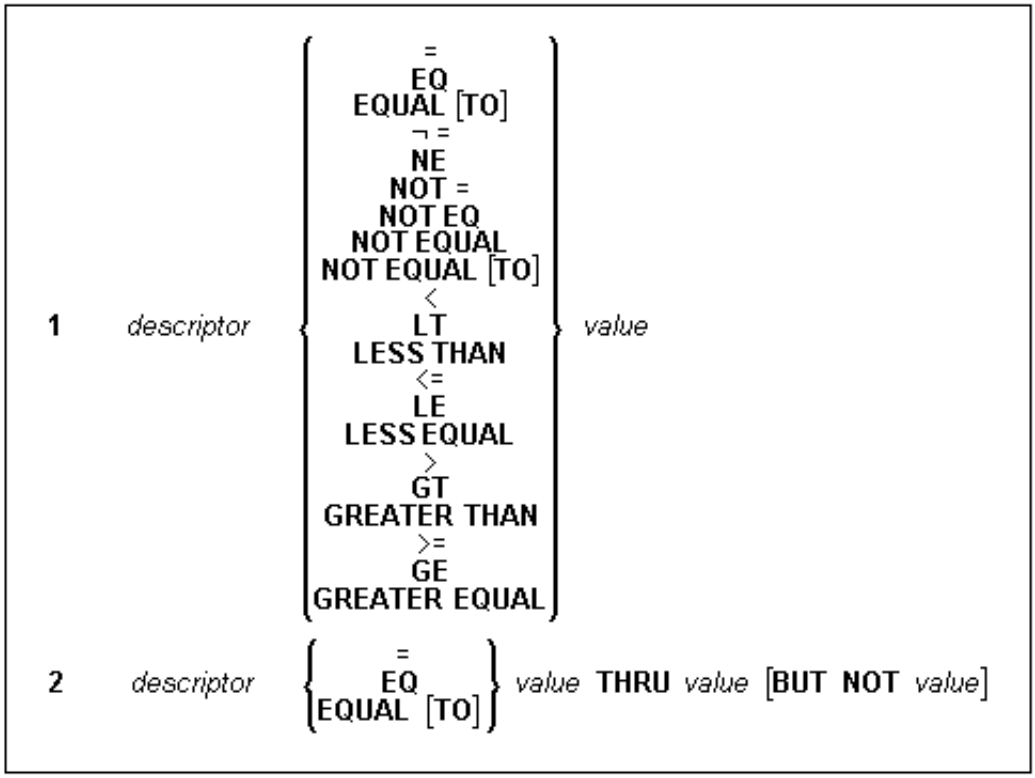
The descriptor must be defined in a VSAM file as a VSAM key field and is marked in the DDM with "P" for primary key or "A" for alternate key.

value

The search value.

The formats of the *descriptor* and the search *value* must be compatible.

Search Criterion for DL/I Files - basic-search-criterion



Operand	Possible Structure					Possible Formats											Referencing Permitted	Dynamic Definition
Descriptor		S	A			A	N	P			B						no	no
Value	C	S				A	N	P			B						yes	no

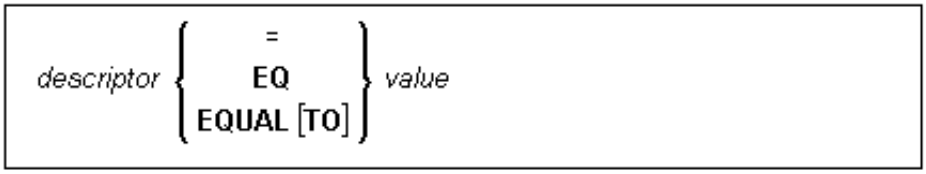
descriptor

The descriptor must be a field defined in DL/I and is marked in the DDM with "D".

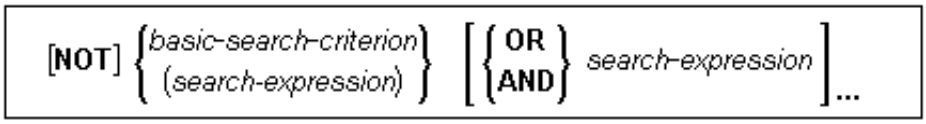
value

The search value.

For HDAM databases, only the following *basic-search-criterion* is possible:



Connecting Search Criteria - for DL/I Files



basic-search-criteria that refer to different segment types must not be connected with the "OR" logical operator.

Examples:

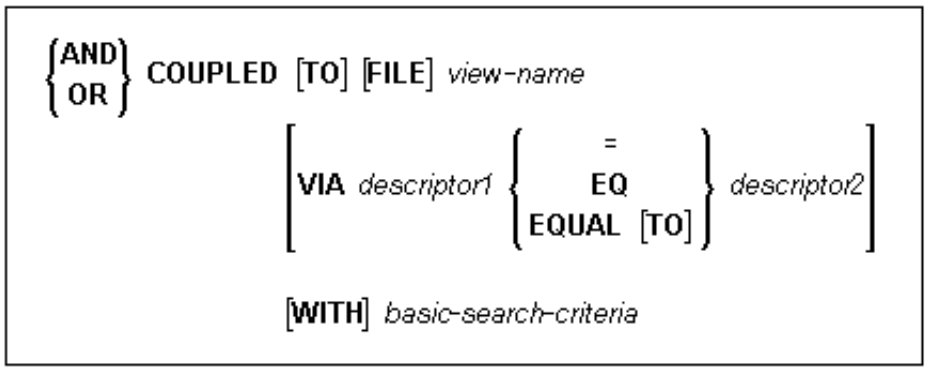
```
FIND COURSE WITH COURSEN > 1
FIND COURSE WITH COURSEN > 1 AND COURSEN < 100
FIND OFFERING WITH (COURSEN-COURSE > 1 OR TITLE-COURSE = 'Natural')
                   AND LOCATION = 'DARMSTADT'
```

Invalid example:

```
FIND OFFERING WITH COURSEN-COURSE > 1 OR LOCATION = 'DARMSTADT'
```

COUPLED-clause

This clause only applies to Adabas databases. This clause is not permitted with Entire System Server.



Operand	Possible Structure					Possible Formats											Referencing Permitted	Dynamic Definition
Descriptor1		S	A			A	N	P			B						no	no
Descriptor2		S	A			A	N	P			B						no	no

Note:
Without the VIA clause, the COUPLED clause may be specified up to 4 times; with the VIA clause, it may be specified up to 42 times.

The COUPLED clause is used to specify a search which involves the use of the Adabas coupling facility. This facility permits database descriptors from different files to be specified in the search criterion of a single FIND statement.

The same Adabas file must not be used in two different FIND COUPLED clauses within the same FIND statement.

A *set-name* (see RETAIN-clause) must not be specified in the *basic-search-criteria*.

Database fields in a file specified within the COUPLED clause are not available for subsequent reference in the program unless another FIND or READ statement is issued separately against the coupled file.

Note:
If the COUPLED clause is used, the main WITH clause may be omitted. If the main WITH clause is omitted, the keywords AND/OR of the COUPLED clause must not be specified.

Physical Coupling without VIA clause

The files used in a COUPLED clause without VIA must be physically coupled using the appropriate Adabas utility (as described in the Adabas documentation).

Example using Physically Coupled Files:

```
/* EXAMPLE 'FNDCPL': FIND (USING COUPLED FILES)
/************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 MAKE
END-DEFINE
/************************************************************
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
                 AND COUPLED TO VEHIC-VIEW WITH MAKE = 'VW'
 DISPLAY NOTITLE NAME
END-FIND
/************************************************************
END
```

The reference to NAME in the DISPLAY statement of the above example is valid since this field is contained in the EMPLOYEES file, whereas a reference to MAKE would be invalid since MAKE is contained in the VEHICLES file, which was specified in the COUPLED clause.

In this example, records will be found only if EMPLOYEES and VEHICLES have been physically coupled.

Logical Coupling - VIA clause

The option "VIA *descriptor1 = descriptor2*" allows you to logically couple multiple Adabas files in a search query. *Descriptor1* is a field from the first view, and *descriptor2* is a field from the second view. The two files need not be physically coupled in Adabas. This COUPLED option uses the soft-coupling feature of Adabas Version 5, as described in the Adabas documentation.

Example using VIA Clause:


```
/* EXAMPLE 'FNSEX1': FIND (USING SOFT COUPLING)
/************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
END-DEFINE
/************************************************************
FIND EMPLOY-VIEW WITH NAME = 'ADKINSON'
  AND COUPLED TO VEHIC-VIEW
    VIA PERSONNEL-ID = PERSONNEL-ID
    WITH MAKE = 'VOLVO'
  DISPLAY PERSONNEL-ID NAME FIRST-NAME
END-FIND
/************************************************************
END
```

```
Page      1                                              91-06-18  14:30:38

  PERSONNEL         NAME              FIRST-NAME
     ID
  --------- -------------------- --------------------

  20011000  ADKINSON             BOB
```

STARTING WITH ISN=operand5

This clause applies only to Adabas and VSAM databases; for VSAM, it is only valid for ESDS.

You can use this clause to specify as *operand5* an Adabas ISN (internal sequence number) or VSAM RBA (relative byte address) respectively, which is to be used as a start value for the selection of records.

This clause may be used for repositioning within a FIND loop whose processing has been interrupted, to easily determine the next record with which processing is to continue. This is particularly useful if the next record cannot be identified uniquely by any of its descriptor values. It can also be useful in a distributed client/server application where the reading of the records is performed by a server program while further processing of the records is performed by a client program, and the records are not processed all in one go, but in batches.

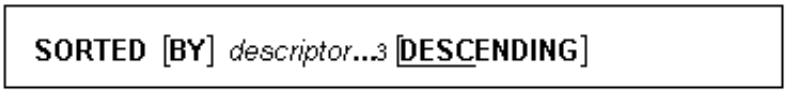
Note:
The start value actually used will not be the value of operand5, but the next higher value.

Example:

See the program FNDSISN in the library SYSEXRM.

SORTED BY-clause

This clause only applies to Adabas and SQL databases.
This clause is not permitted with Entire System Server.



The SORTED BY clause is used to cause Adabas to sort the selected records based on the sequence of one to three descriptors. The descriptors used for controlling the sort sequence may be different from those used for selection.

By default, the records are sorted in *ascending* sequence of values; if you want them to be in descending sequence, specify the keyword DESCENDING. The sort is performed using the Adabas inverted lists and does not result in any records being read.

Note:
The use of this clause may result in significant overhead if any descriptor used to control the sort sequence contains a large number of values. This is because the entire value list may have to be scanned until all selected records have been located in the list. When a large number of records is to be sorted, you should use the SORT statement.

Adabas sort limits (see the ADARUN LS parameter in the Adabas documentation) are in effect when the SORTED BY clause is used.

A descriptor which is contained in a periodic group must not be specified in the SORTED BY clause. A multiple-value field (without an index) may be specified.

Except on OpenVMS and mainframe computers, non-descriptors may also be specified in the SORTED BY clause.

If the SORTED BY clause is used, the RETAIN clause must not be used.

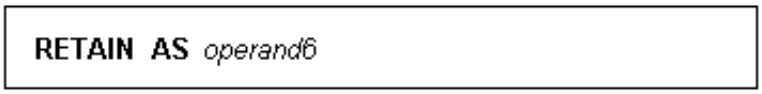
Example of SORTED BY Clause:

```
/* EXAMPLE 'FNDSOR': FIND (SORTED BY CLAUSE)
/*****************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 CITY
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
END-DEFINE
/*****************************************************
LIMIT 10
FIND EMPLOY-VIEW WITH CITY = 'FRANKFURT'
                 SORTED BY NAME PERSONNEL-ID
 DISPLAY NOTITLE NAME (IS=ON) FIRST-NAME PERSONNEL-ID
END-FIND
/*****************************************************
END
```


NAME	FIRST-NAME	PERSONNEL ID
BAECKER	JOHANNES	11500345
BECKER	HERMANN	11100311
BERGMANN	HANS	11100301
BLAU	SARAH	11100305
BLOEMER	JOHANNES	11200312
DIEDRICHS	HUBERT	11600301
DOLLINGER	MARGA	11500322
FALTER	CLAUDIA	11300311
	HEIDE	11400311
FREI	REINHILD	11500301

RETAIN-clause

This clause only applies to Adabas databases. This clause is not permitted with Entire System Server.



Operand	Possible Structure		Possible Formats	Referencing Permitted	Dynamic Definition
Operand6	C	S	A	yes	no

By using the RETAIN clause, the result of an extensive search in large files can be retained for further processing. The selection is retained as an "ISN-set" in the Adabas work file. The set may be used in subsequent FIND statements as a basic search criterion for further refinement of the set or for further processing of the records. The set created is file-specific and may only be used in another FIND statement that processes the same file. The set may be referenced by any Natural program.

Set Name - operand6

The set name is used to identify the record set. It may be specified as an alphanumeric constant or as the content of an alphanumeric user-defined variable. Duplicate set names are not checked; consequently, if a duplicate set name is specified, the new set replaces the old set.

Releasing Sets

There is no specific limit for the number of sets that can be retained or the number of ISNs in a set. It is recommended that the minimum number of ISN sets needed at one time be defined. Sets that are no longer needed should be released using the RELEASE SETS statement.

If they are not released with a RELEASE statement, retained sets exist until the end of the Natural session, or until a logon to another library, when they are released automatically. A set created by one program may be referenced by another program for processing or further refinement using additional search criteria.

Updates by Other Users

The records identified by the ISNs in a retained set are not locked against access and/or update by other users. Before you process records from the set, it is therefore useful to check whether the original search criteria which were used to create the set are still valid: This check is done with another FIND statement, using the set name in the WITH clause as basic search criterion and specifying in a WHERE clause the original search criterion (that is, the basic search criteria as specified in the WITH clause of the FIND statement which was used to create the set).

Restriction

If the RETAIN clause is used, the SORTED BY clause must not be used.

Example of a RETAIN Clause:

```
 * EXAMPLE '': FIND (RETAIN CLAUSE) AND RELEASE
************************************************
DEFINE DATA LOCAL
 1 EMPLOY-VIEW VIEW OF EMPLOYEES
   2 CITY
   2 BIRTH
   2 NAME
 1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19400101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH BIRTH GT #BIRTH
                        RETAIN AS 'AGESET1'
IF *NUMBER = 0
  STOP
END-IF
*
FIND EMPLOY-VIEW WITH 'AGESET1' AND CITY = 'NEW YORK'
  DISPLAY NOTITLE NAME CITY BIRTH (EM=YYYY-MM-DD)
END-FIND
*
RELEASE SET 'AGESET1'
END
```

```
        NAME                 CITY            DATE
                                              OF
                                             BIRTH
-------------------- -------------------- ----------

RUBIN                NEW YORK             1945-10-27
WALLACE              NEW YORK             1945-08-04
```

WHERE-clause

WHERE *logical-condition*

The WHERE clause is used to specify an additional selection criterion which is evaluated *after* a record (selected with the WITH clause) has been read and *before* any processing is performed on the record (including AT BREAK evaluation).

The syntax for a *logical-condition* is described in the section Logical Condition Criteria in the Natural Reference documentation.

If a processing limit is specified in a FIND statement containing a WHERE clause, records which are rejected as a result of the WHERE clause are *not* counted against the limit. These records are, however, counted against any global limit specified in a Natural session parameter, the GLOBALS command, or LIMIT statement.

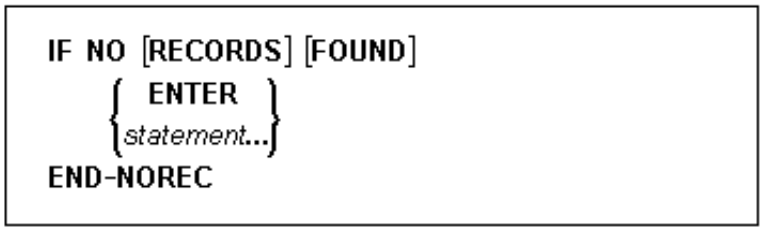
Example of WHERE Clause:

```
/* EXAMPLE 'FNDWHE': FIND (WHERE CLAUSE)
/*********************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 JOB-TITLE
  2 CITY
END-DEFINE
/*********************************************************
FIND EMPLOY-VIEW WITH CITY = 'PARIS'
                 WHERE JOB-TITLE = 'INGENIEUR COMMERCIAL'
 DISPLAY NOTITLE CITY JOB-TITLE PERSONNEL-ID NAME
END-FIND
/*********************************************************
END
```

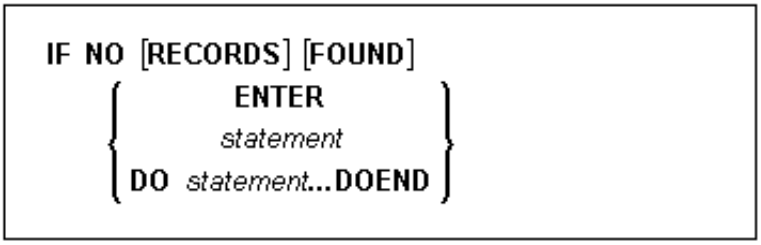
```
      CITY                  CURRENT          PERSONNEL         NAME
                             POSITION             ID
------------------- ------------------------ --------- --------------------

PARIS               INGENIEUR COMMERCIAL      50007300  CAHN
PARIS               INGENIEUR COMMERCIAL      50006500  MAZUY
PARIS               INGENIEUR COMMERCIAL      50004400  VALLY
PARIS               INGENIEUR COMMERCIAL      50002800  BRETON
PARIS               INGENIEUR COMMERCIAL      50001000  GIGLEUX
```

IF NO RECORDS FOUND-clause

Structured Mode Syntax



Reporting Mode Syntax



The IF NO RECORDS FOUND clause may be used to cause a processing loop initiated with a FIND statement to be entered in the event that no records meet the selection criteria specified in the WITH and WHERE clauses.

If no records meet the specified WITH and WHERE criteria, the IF NO RECORDS FOUND clause causes the FIND processing loop to be executed once with an "empty" record. If this is not desired, specify the statement ESCAPE BOTTOM within the IF NO RECORDS FOUND clause.

If one or more statements are specified with the IF NO RECORDS FOUND clause, the statements will be executed immediately before the processing loop is entered. If no statements are to be executed before entering the loop, the keyword ENTER must be used.

Database Values

Unless other value assignments are made in the statements accompanying an IF NO RECORDS FOUND clause, Natural will reset to empty all database fields which reference the file specified in the current loop.

Evaluation of System Functions

Natural system functions are evaluated once for the empty record that is created for processing as a result of the IF NO RECORDS FOUND clause.

Restriction

This clause cannot be used with FIND FIRST, FIND NUMBER and FIND UNIQUE.

Example of IF NO RECORDS FOUND Clause:


```
/* EXAMPLE 'FNDIFN': FIND (IF NO RECORDS FOUND CLAUSE)
/************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/************************************************************
LIMIT 15
EMP. READ EMPLOY-VIEW BY NAME STARTING FROM 'JONES'
/************************************************************
VEH.  FIND VEHIC-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (EMP.)
       IF NO RECORDS FOUND
        MOVE '*** NO CAR ***' TO MAKE
       END-NOREC
       DISPLAY NOTITLE
          NAME (EMP.) (IS=ON) FIRST-NAME (EMP.) (IS=ON) MAKE (VEH.)
      END-FIND
/************************************************************
     END-READ
END
```

```
      NAME              FIRST-NAME             MAKE
-------------------- -------------------- --------------------

JONES                VIRGINIA             CHRYSLER
                     MARSHA               CHRYSLER
                                          CHRYSLER
                     ROBERT               GENERAL MOTORS
                     LILLY                FORD
                                          MG
                     EDWARD               GENERAL MOTORS
                     MARTHA               GENERAL MOTORS
                     LAUREL               GENERAL MOTORS
                     KEVIN                DATSUN
                     GREGORY              FORD
JOPER                MANFRED              *** NO CAR ***
JOUSSELIN            DANIEL               RENAULT
JUBE                 GABRIEL              *** NO CAR ***
JUNG                 ERNST                *** NO CAR ***
JUNKIN               JEREMY               *** NO CAR ***
KAISER               REINER               *** NO CAR ***
```

System Variables with the FIND Statement

The Natural system variables *ISN, *NUMBER, and *COUNTER are automatically created for each FIND statement issued. A reference number must be supplied if the system variable was referenced outside the current processing loop or through a FIND UNIQUE, FIND FIRST, or FIND NUMBER statement. The format/length of each of these system variables is P10; this format/length cannot be changed.

*ISN

For Adabas databases, *ISN contains the Adabas internal sequence number (ISN) of the record currently being processed. *ISN is not available for the FIND NUMBER statement.

For VSAM databases, *ISN contains the relative byte address (RBA) of the record currently being processed (ESDS files only).

For DL/I and SQL databases, and with Entire System Server, *ISN is not available.

*NUMBER

Contains the number of records which satisfied the basic search criterion specified in the WITH clause.

For DL/I databases, *NUMBER contains "0" if no segment occurrences satisfy the search criterion, and a value of "9999" if at least one segment occurrence satisfies the search criterion.

For VSAM databases, *NUMBER only contains a meaningful value if the EQUAL TO operator is used in the search criterion. With any other operator, *NUMBER will be "0" if no records have been found; any other value indicates that records have been found, but the value will have no relation to the number of records actually found.

For SQL databases, see the section System Variables in the Natural Reference documentation.

With Entire System Server, *NUMBER is not available.

*COUNTER

Contains the number of times the processing loop has been entered.

Example using System Variables:

```
/* EXAMPLE 'FNDVAR': FIND (SYSTEM VARIABLES *ISN, *NUMBER, *COUNTER)
/************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 CITY
END-DEFINE
/************************************************************************
LIMIT 3
FIND EMPLOY-VIEW WITH CITY = 'MADRID'
 DISPLAY NOTITLE PERSONNEL-ID NAME
                 *ISN *NUMBER *COUNTER
END-FIND
/************************************************************************
END
```

```
PERSONNEL         NAME           ISN       NMBR      CNT
    ID
 --------- -------------------- --------- --------- ---------

 60000114  DE JUAN                   401        41         1
 60000136  DE LA MADRID              402        41         2
 60000209  PINERO                    406        41         3
```

Multiple FIND Statements

Multiple FIND statements may be issued to create nested loops whereby an inner loop is entered for each record selected in the outer loop.

Example of Multiple FIND Statements:

In the following example, first all people named SMITH are selected from the EMPLOYEES file. Then the PERSONNEL-ID from the EMPLOYEES file is used as the search key for an access to the VEHICLES file. The resulting report shows the NAME and FIRST-NAME (obtained from the EMPLOYEES file) of all people named SMITH as well as the MAKE of each car (obtained from the VEHICLES file) owned by these people:

```
/* EXAMPLE 'FNDMUL': FIND (USING MULTIPLE FILES)
/***********************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 PERSONNEL-ID
  2 NAME
  2 FIRST-NAME
1 VEHIC-VIEW VIEW OF VEHICLES
  2 PERSONNEL-ID
  2 MAKE
END-DEFINE
/***********************************************************************
LIMIT 15
EMP. FIND EMPLOY-VIEW WITH NAME = 'SMITH'
VEH.  FIND VEHIC-VIEW WITH PERSONNEL-ID = EMP.PERSONNEL-ID
        IF NO RECORDS FOUND
         MOVE '*** NO CAR ***' TO MAKE
        END-NOREC
        DISPLAY NOTITLE
            EMP.NAME (IS=ON)
            EMP.FIRST-NAME (IS=ON) VEH.MAKE
      END-FIND
    END-FIND
/***********************************************************************
END
```

```
      NAME              FIRST-NAME             MAKE
-------------------- -------------------- --------------------

SMITH                GERHARD              ROVER
                     SEYMOUR              *** NO CAR ***
                     MATILDA              FORD
                     ANN                  *** NO CAR ***
                     TONI                 TOYOTA
                     MARTIN               *** NO CAR ***
                     THOMAS               FORD
                     SUNNY                *** NO CAR ***
                     JUNE                 JAGUAR
                     MARK                 FORD
                     LOUISE               CHRYSLER
                     MAXWELL              MERCEDES-BENZ
                                          MERCEDES-BENZ
                     ELSA                 CHRYSLER
                     CHARLY               CHRYSLER
                     LEE                  *** NO CAR ***
```

FIND FIRST

The FIND FIRST statement may be used to select and process the first record which meets the WITH and WHERE criteria.

For Adabas databases, the record processed will be the record with the lowest Adabas ISN from the set of qualifying records.

This statement does *not* initiate a processing loop.

Restrictions

FIND FIRST can only be used in reporting mode.

FIND FIRST is not available for DL/I and SQL databases.

The IF NO RECORDS FOUND clause must not be used in a FIND FIRST statement.

System Variables with FIND FIRST

The following Natural system variables are available with the FIND FIRST statement:

*ISN

Contains the Adabas ISN of the selected record. *ISN will be "0" if no record is found after the evaluation of the WITH and WHERE criteria.

*ISN is not available for VSAM databases or with Entire System Server.

*NUMBER

Contains the number of records found after the evaluation of the WITH criterion and before evaluation of any WHERE criterion. *NUMBER will be "0" if no record meets the WITH criterion.

*NUMBER is not available with Entire System Server.

*COUNTER

Contains "1" if a record was found; contains "0" if no record was found.

Example of FIND FIRST

See the program FNDFIR in the library SYSEXRM.

FIND NUMBER

The FIND NUMBER statement is used to determine the number of records which satisfy the WITH/WHERE criteria specified. It does *not* result in the initiation of a processing loop and *no data fields from the database are made available*.

Note:
Use of the WHERE clause may result in significant overhead.

Restrictions

The SORTED BY and IF NO RECORDS FOUND clauses must not be used with the FIND NUMBER statement.

The WHERE clause cannot be used in structured mode.

FIND NUMBER is not available for DL/I databases.

FIND NUMBER is not available with Entire System Server.

System Variables with FIND NUMBER

The following Natural system variables are available with the FIND NUMBER statement:

*NUMBER

Contains the number of records found after the evaluation of the WITH criterion.

*COUNTER

Contains the number of records found after the evaluation of the WHERE criterion.

*COUNTER is only available if the FIND NUMBER statement contains a WHERE clause.

Example of FIND NUMBER:

```
 * EXAMPLE '': FIND NUMBER
*******************************
DEFINE DATA LOCAL
 1 EMPLOY-VIEW VIEW OF EMPLOYEES
   2 CITY
   2 BIRTH
 1 #BIRTH (D)
END-DEFINE
*
MOVE EDITED '19500101' TO #BIRTH (EM=YYYYMMDD)
*
FIND NUMBER EMPLOY-VIEW WITH CITY = 'MADRID'
                        WHERE BIRTH LT #BIRTH
*
WRITE NOTITLE 'TOTAL RECORDS SELECTED:       ' *NUMBER
            / 'TOTAL BORN BEFORE 1 JAN 1950: ' *COUNTER
*
END
```

```
TOTAL RECORDS SELECTED:                 41
TOTAL BORN BEFORE 1 JAN 1950:           16
```

FIND UNIQUE

The FIND UNIQUE statement may be used to ensure that only one record is selected for processing. It does not result in the initiation of a processing loop. If a WHERE clause is specified, an automatic internal processing loop is created to evaluate the WHERE clause.

If no records or more than one record satisfy the criteria, an error message will be issued. This condition can be tested with the ON ERROR statement.

Restrictions

FIND UNIQUE can only be used in reporting mode.

FIND UNIQUE is not available for DL/I databases or with Entire System Server.

For SQL databases, FIND UNIQUE cannot be used. (Exception: On mainframe computers, FIND UNIQUE can be used for primary keys; however, this is only permitted for compatibility reasons and should not be used.)

The SORTED BY and IF NO RECORDS FOUND clauses must not be used with the FIND UNIQUE statement.

Example of FIND UNIQUE

See the program FNDUNQ in the library SYSEXRM.

